

SciCat Performance Assessment

SciCatCon 2025

Igor Khokhriakov aka Ingvord

MORE DETAILS IN THE DAPHNE4NFDI ANNUAL MEETING POSTER



<https://zenodo.org/records/15068411>

[Communities](#)
[My dashboard](#)

Published March 22, 2025 | Version v1

Poster Open

Benchmarking SciCat Performance: Data-In & Data-Out for Beamline Operations .

Khokhriakov, Igor (Project leader)¹

Contributors

Others: Hinzmann, Regina¹ ; Parthasarathy Tirumalai Nallam Chakravarty¹

This work was partially supported by the consortium DAPHNE4NFDI in the context of the work of the NFDI e.V. The consortium is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - project number 460248799.

Files

[SciCat_Benchmarking_Poster.pdf](#)

1 of 1

Automatic Zoom

Data-In & Data-Out for Beamline Operations.

Authors: Igor Khokhriakov, Parthasarathy Tirumalai Nallam Chakravarty (FS-SC)
Regina Hinzmann (IT-INFA)

Problem Statement

Beamline operations generate massive amounts of data that need to be efficiently ingested, stored, and retrieved for analysis. An experimental setup requires more complex data storage and retrieval systems, creating performance challenges for data management systems like SciCat.

Key questions we aim to answer:

- Can SciCat handle real-world scientific data loads efficiently?
- How does metadata complexity impact performance?
- Does increasing request rates improve or degrade performance?
- What happens when the dataset size grows?
- How can beamline scientists optimize data workflows?

Understanding SciCat's scalability and performance limits is critical for ensuring smooth beamline operations, optimizing data processing pipelines, and improving real-time data availability for scientific analysis.

Methodology

To evaluate SciCat's ability to handle large-scale scientific data, we conducted two types of performance tests:

- Data Retrieval (Read Performance)**
 - Queried SciCat by PID (PhotonIdentifier) to simulate typical scenarios in beamline workflows.

Results

Our benchmarking experiments reveal key insights into SciCat's performance under different loads for both data retrieval (read) and data ingestion (write).

- Data Retrieval (Looking for Data by PID):** Performance remains stable at default 400 requests. Query times are relatively constant across 1K, 10K, 100K, and 1M dataset sizes.
- Write (Ingestion):** Low request rates (R10, R100) show minimal performance degradation, while higher rates (R1000) cause noticeable increases in latency.
- Dataset Size Impact:** The 100K dataset anomaly observed in an earlier test was due to an unoptimized pipeline, reflecting the importance of pipeline refinement.
- Scaling Impact:** Reducing dataset count (N=10) improved performance compared to previous tests with higher concurrency.
- Conclusion:** SciCat's query operations scale reasonably well but can be impacted by high concurrency or unoptimized query patterns.

- Data Ingestion (Uploading Datasets with Different Metadata Complexity)** Metadata complexity significantly impacts ingestion time. Datasets with 100 metadata fields ingest efficiently at moderate rates. 250 metadata fields show latency spikes at higher request rates. 1000 metadata fields result in severe ingestion delays, making SciCat unsuitable for high-volume data ingestion in this scenario.
- Request rate impact:** At low rates (R10, R100), ingestion remains manageable across all metadata complexities. At R1000 and R10000, ingestion latency increases drastically, exceeding 1 minute per request.
- Conclusion:** SciCat ingestion does not scale well under high metadata complexity. Strategies such as data pre-processing, metadata optimization, or asynchronous ingestion pipelines are necessary for high-throughput experiments.

4 VIEWS

16 DOWNLOADS

Show more details

Versions

Version v1 Mar 22, 2025

10.5281/zenodo.15068411

Cite all versions? You can cite all versions by using the DOI 10.5281/zenodo.15068410. This DOI represents all versions, and will always resolve to the latest one. Read more.

External resources

Indexed in

OpenAIRE

Communities

DAPHNE4NFDI

Details

DOI [10.5281/zenodo.15068411](https://doi.org/10.5281/zenodo.15068411)

Resource type Poster

Publisher Zenodo

Conference

DAPHNE4NFDI Annual Meeting 2025, Berlin, 21-26 March 2025

SciCat, part 1

Data-Out, PID

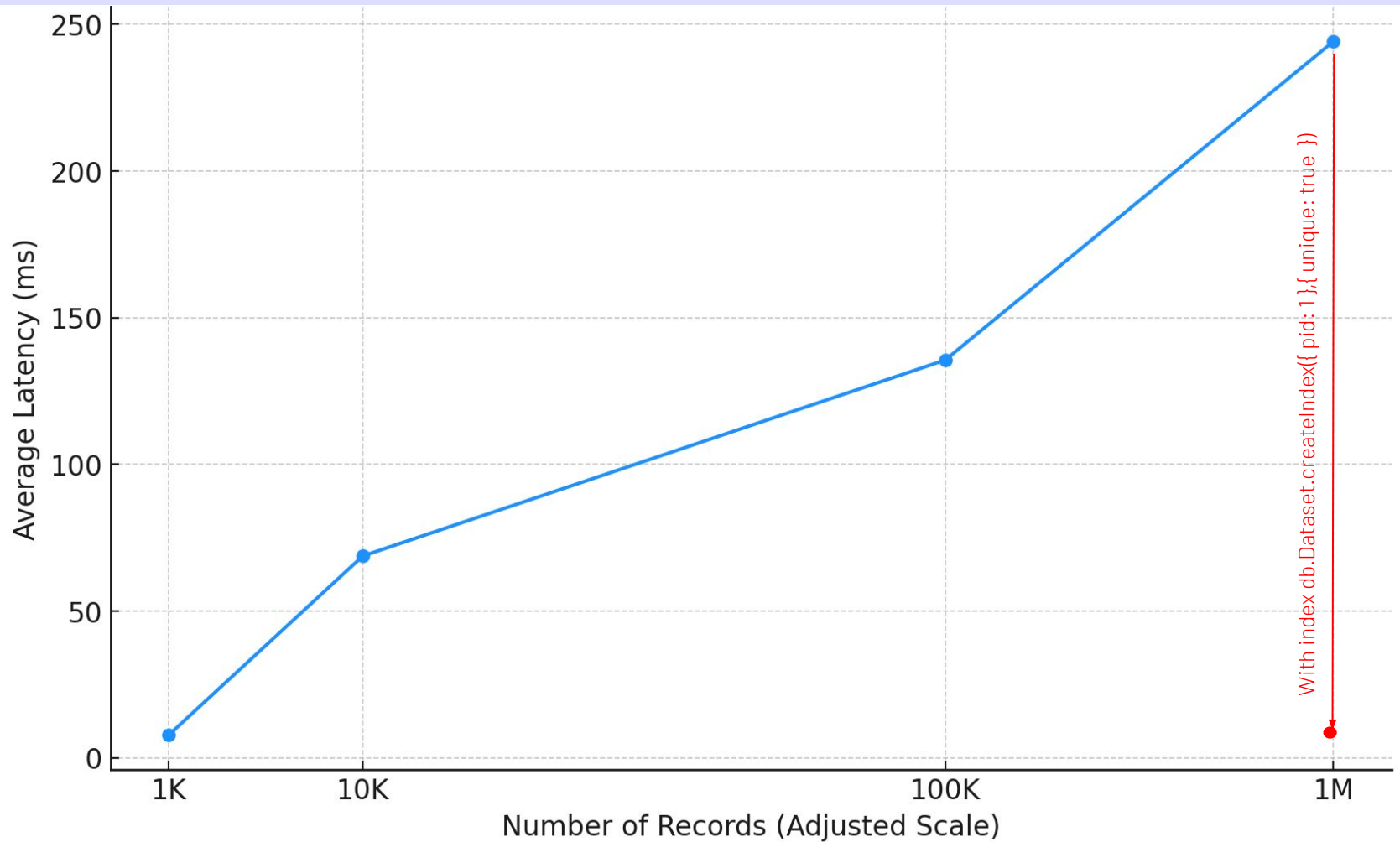
SciCatCon 2025

Igor Khokhriakov aka Ingvord

QUIZ

What result do you expect?

Average Latency VS Number of Records in the DB (C10, R1000)



METHODOLOGY

Testing Tool:

- The benchmarking was conducted using [wrk](#), a modern HTTP benchmarking tool capable of generating significant load with multithreading and connection options.

Test Scenarios:

- Multiple configurations were tested by varying:
 - **Dataset size:** 1K, 10K, 100K and 1M records.
 - **Request rate:** R10, R100, and R1000 requests per second.
 - **Concurrent connections:** 50 and 100 connections.

Endpoints:

- The REST API endpoint [/datasets/fullquery](#) was tested with query parameters to simulate real-world search patterns.

Metrics Collected:

- **Latency:** Average, standard deviation, and percentiles (50th, 75th, 90th, 99th).
- **Throughput:** Achieved requests per second and total requests processed.
- **Reliability:** Number of socket errors (timeouts, connection issues).

1_000 Records, 10 Clients, Rate {10..1000}

Configuration	Avg Latency	50th %ile Latency	99th %ile Latency	Requests/sec	Timeouts
R1000	9.83ms	4.78ms	163.97ms	1000.01	-
R100	12.06ms	10.81ms	30.91ms	100.05	-
R10	12.64ms	11.93ms	29.22ms	10.05	-

Conclusions

- **With 1K records, the system is highly efficient at all request rates, even at R1000, where it maintains excellent performance.**
- **Limiting clients to 10 (-c10) ensures stable performance without significant latency spikes, even under high throughput.**
- **Compared to 10K and 100K records, 1K datasets show minimal impact from increasing request rates, indicating the backend can efficiently handle smaller datasets at scale.**

10_000 Records, 10 Clients, Rate {10..1000}

Configuration	Avg Latency	50th %ile Latency	99th %ile Latency	Requests/sec	Timeouts
R1000	51.23ms	7.22ms	1.13s	999.73	-
R100	20.12ms	15.59ms	179.33ms	100.02	-
R10	16.80ms	16.42ms	36.13ms	10.06	-

Conclusions

- Limiting clients to 10 results in significantly improved system performance at all load levels.
- Unlike 100K records, where R1000 introduced substantial latency spikes, 10K records show no major degradation even at R1000.
- The system scales well with 10K records, achieving the target request rate with low latency.
- For real-world usage, keeping concurrency controlled (e.g., -c10) can prevent unnecessary queuing delays and improve stability.

100_000 Records, 10 Clients, Rate {10..1000}

Configuration	Avg Latency	50th %ile Latency	99th %ile Latency	Requests/sec	Timeouts
R1000	4.36s	2.91s	23.84s	971.41	-
R100	93.13ms	12.64ms	2.59s	98.53	-
R10	23.06ms	13.51ms	241.54ms	10.02	-

Conclusions

- Lowering concurrent clients (-c10) significantly reduces queuing effects and improves system stability.
- The system **scales well** at low to moderate request rates (R10, R100) but **still struggles** under R1000, with **high average and 99th percentile latencies**.
- Compared to previous high concurrency tests, reducing client count has a major positive impact on performance.

1_000_000 Records, 10 Clients, Rate {10..1000}

Configuration	Avg Latency	50th %ile Latency	99th %ile Latency	Requests/sec	Timeouts
R1000	127.60ms	6.82ms	2.08s	997.52	-
R100	14.49ms	11.60ms	74.30ms	100.06	-
R10	17.87ms	15.08ms	148.48ms	10.02	-

Conclusions

- Lowering concurrent clients (-c10) significantly reduces queuing effects and improves system stability.
- The system scales well at low to moderate request rates (R10, R100) but still struggles under R1000, with high average and 99th percentile latencies.
- Compared to previous high concurrency tests, reducing client count has a major positive impact on performance.

Combined Results Tableview

[illegible]

CONCLUSIONS

- **System scales well under high request rates**, with **1K, 10K, 100K, and 1M records** maintaining low latency at **R1000**.
- **100K records initially showed a performance spike**, but this was due to **system overload (100% RAM and swap usage)**—when properly tested, **it performed reasonably well**.
- **Limiting concurrency (-c10) significantly improves stability**, reducing queuing effects and keeping latency distribution tighter.
- **1M records perform better than expected**, suggesting **efficient caching, indexing, or database optimizations at scale**.
- **Low (R10) and moderate (R100) request rates show excellent stability**, with **99th percentile latency** remaining within acceptable ranges.
- **At high load (R1000), 99th percentile latencies increase but remain within operational limits (~2s for 1M records)**.
- **Final takeaway**: The system handles high request loads efficiently across dataset sizes, and the **previous 100K anomaly serves as an anchor for discussions on system health monitoring**.

- Optimize database indexing and query execution plans for large datasets.
- Implement caching mechanisms for frequently accessed data.
- Conduct capacity planning to identify and address resource bottlenecks.
- Consider load balancing strategies to handle higher concurrency and throughput.
- Finally be ready to auto-scale the deployment.

SIDE NOTES



Search or jump to...

ctrl+k



Home > Dashboards > Node Exporter Full



Share

Edit



Datasource default Job node Host localhost:9100

GitHub

Grafana

Last 15 minutes



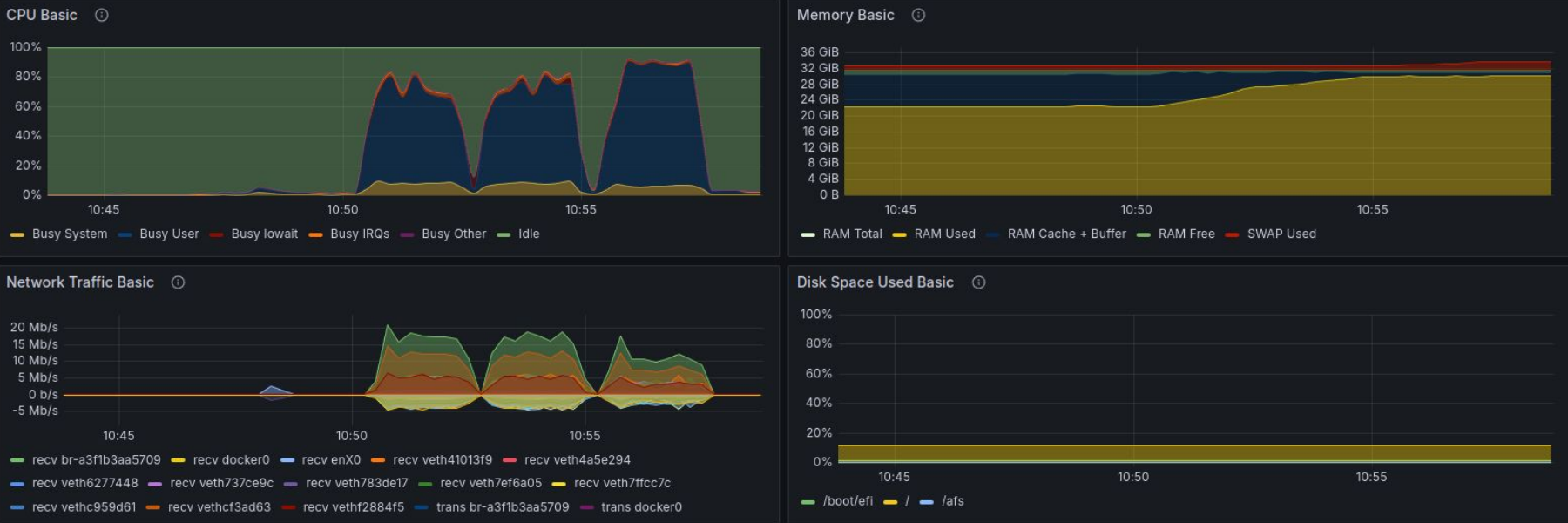
5s

Auto

Quick CPU / Mem / Disk



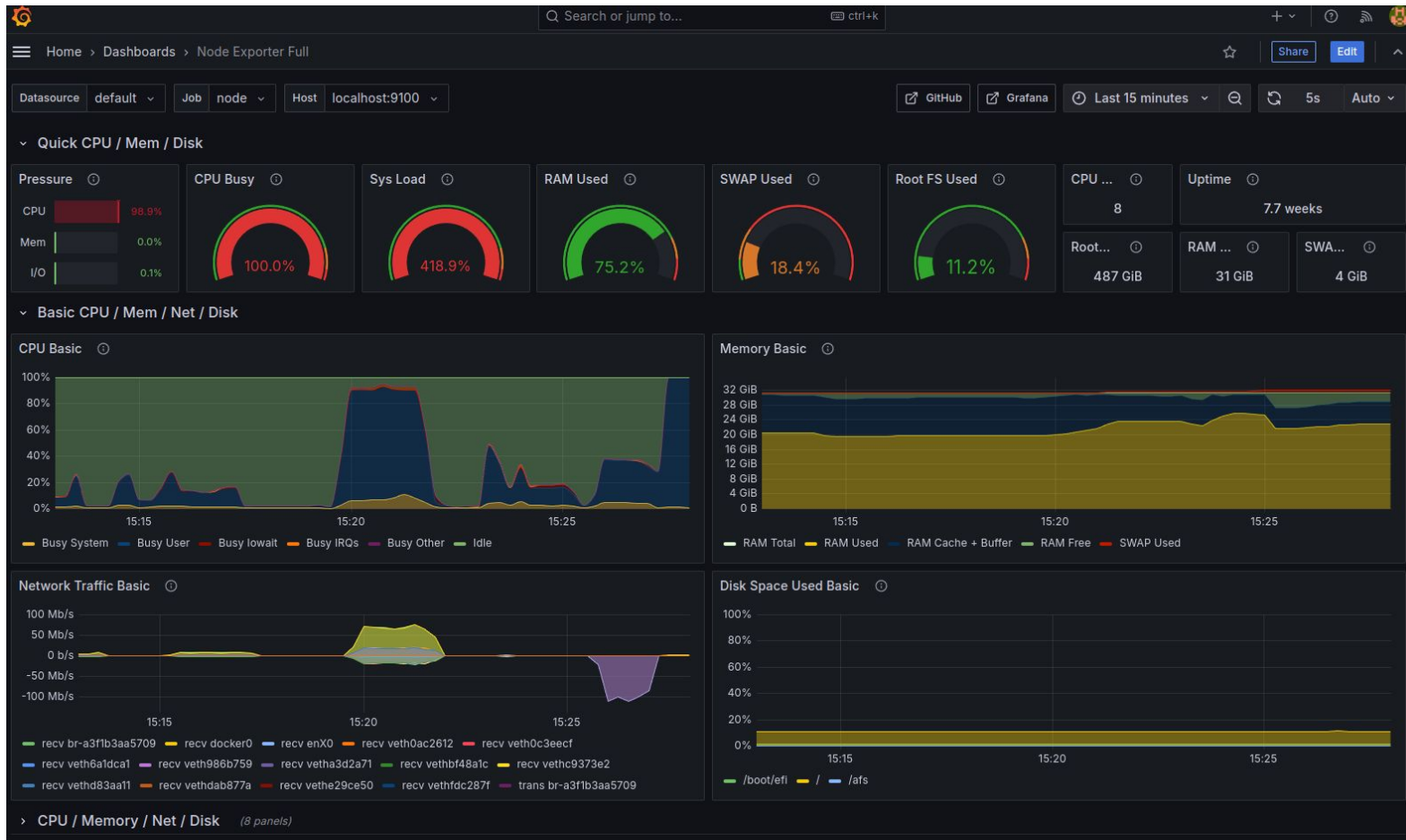
Basic CPU / Mem / Net / Disk



> CPU / Memory / Net / Disk (8 panels)

SYSTEM LOAD 100_000, c=100, R=1000

16

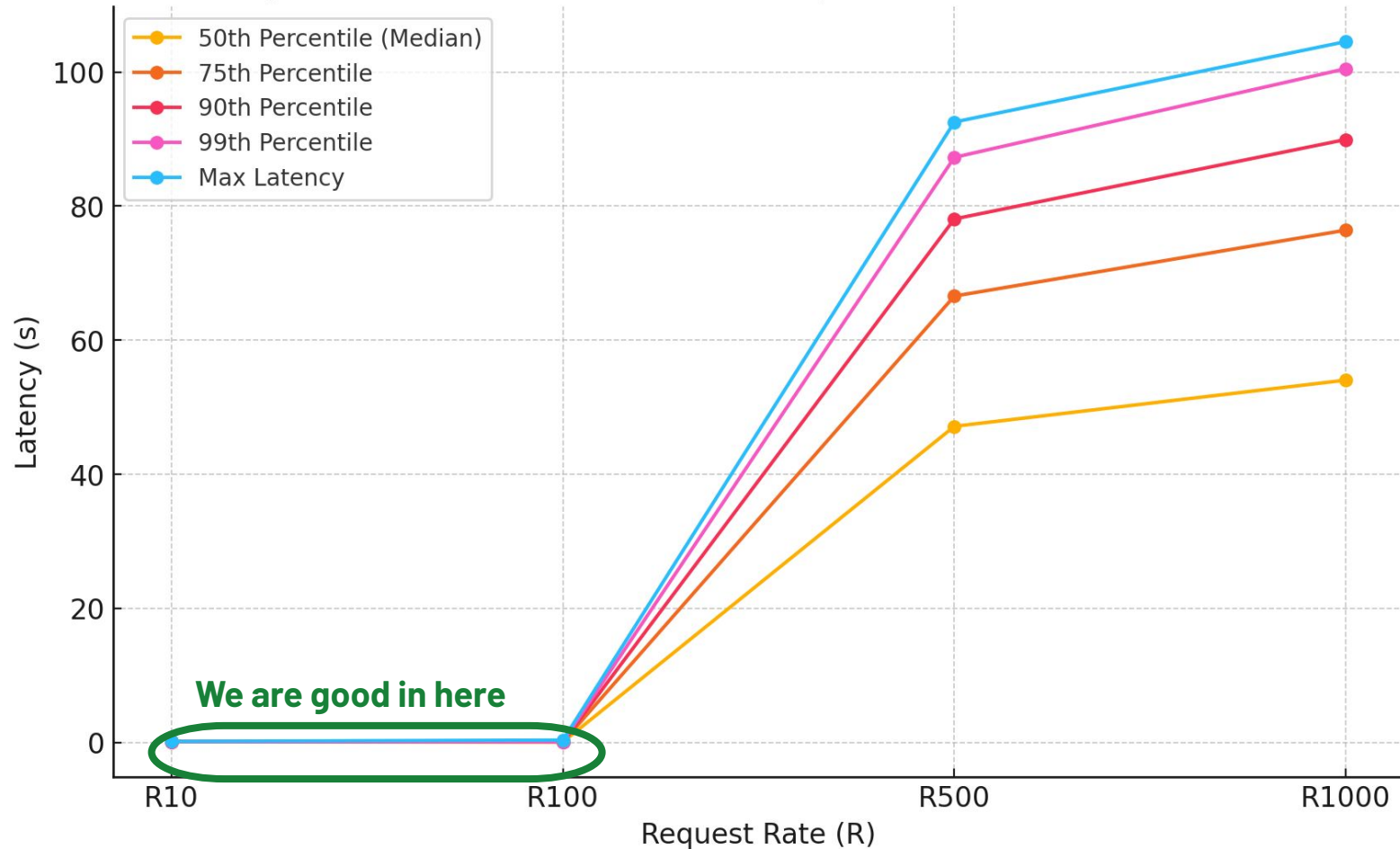


SciCat, part 2

Data-In, Metadata

Igor Khokhriakov aka Ingvord

Latency Distribution Across Different Request Rates (100 Metadata Fields)



CONCLUSIONS

Summary of Ingestion Performance Tests (100 Metadata Fields)

✓ Stable performance at low request rates (R10, R100)

- Low latency (50th percentile: 86ms at R10, 65ms at R100).
- The system handles these loads efficiently without degradation.

✗ Severe performance degradation at higher loads (R500, R1000)

- 50th percentile latency jumps to 47s (R500) and 54s (R1000).
- 99th percentile reaches 87s (R500) and over 100s (R1000), making real-time ingestion infeasible.

✗ Backend reaches a hard capacity limit (~140-150 req/sec)

- At R500 and R1000, the system only achieves ~140-150 req/sec, far below the target rates.
- Indicates queueing, serialization, or database bottlenecks.

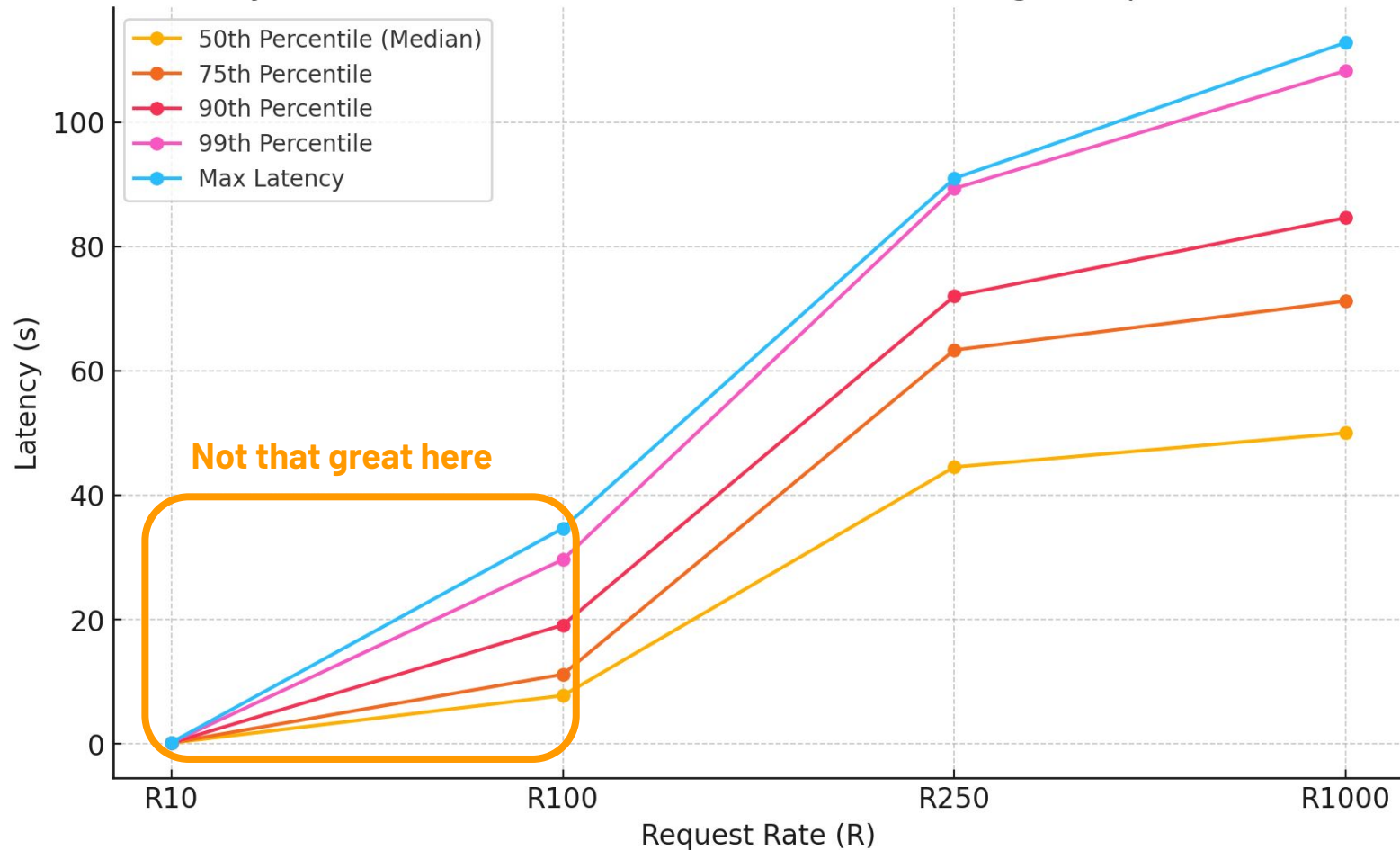
✓ No outright failures, but long processing delays

- No timeouts or rejected requests, meaning the system is processing all requests but at a slow pace.

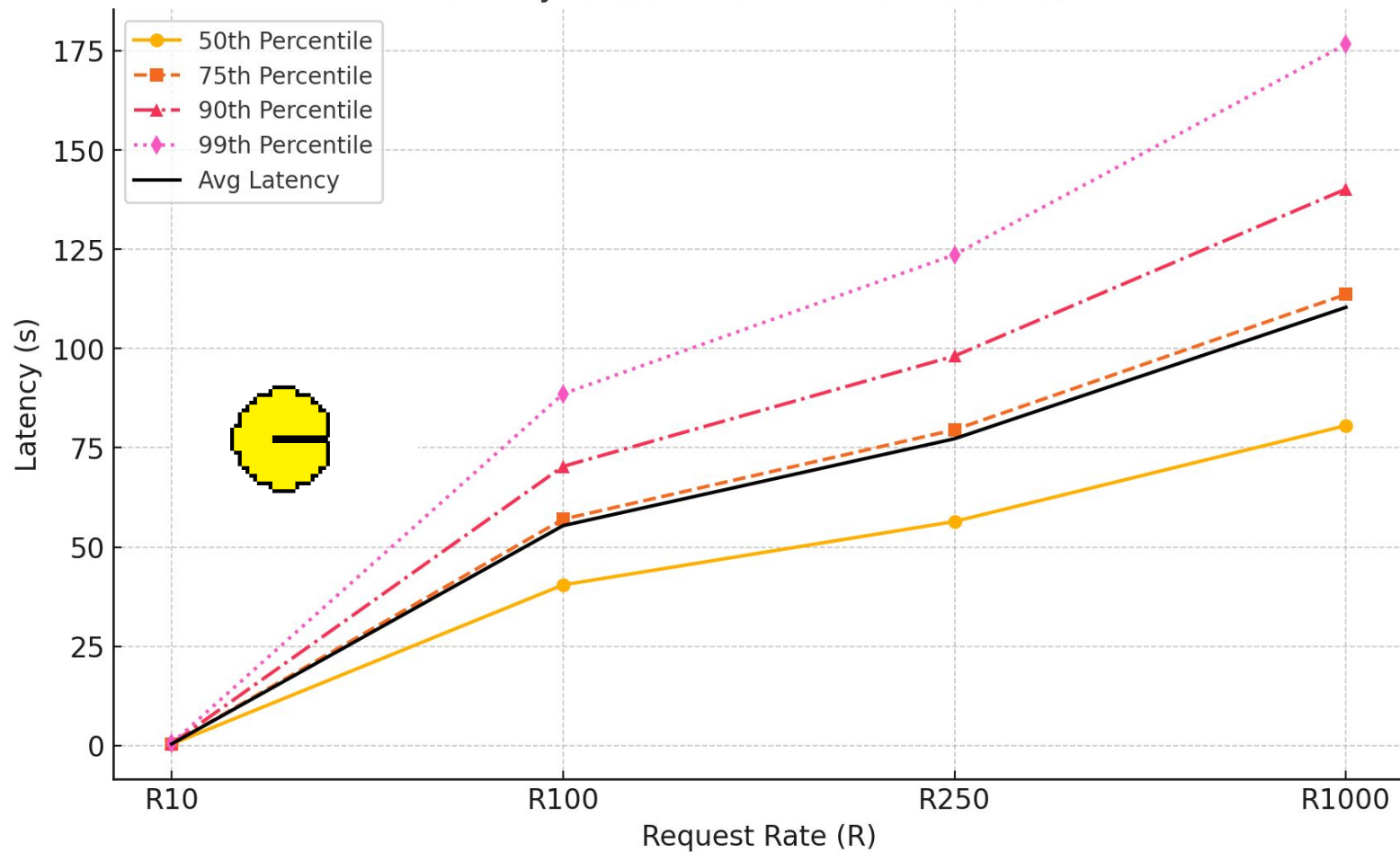
✗ Lack of horizontal scalability observed

- Higher concurrency does not improve throughput.
- Suggests database, transaction, or resource contention issues.

Latency Distribution for 250 Metadata Fields (Including Extrapolated R1000)



Latency Trends for 1000 Metadata Fields





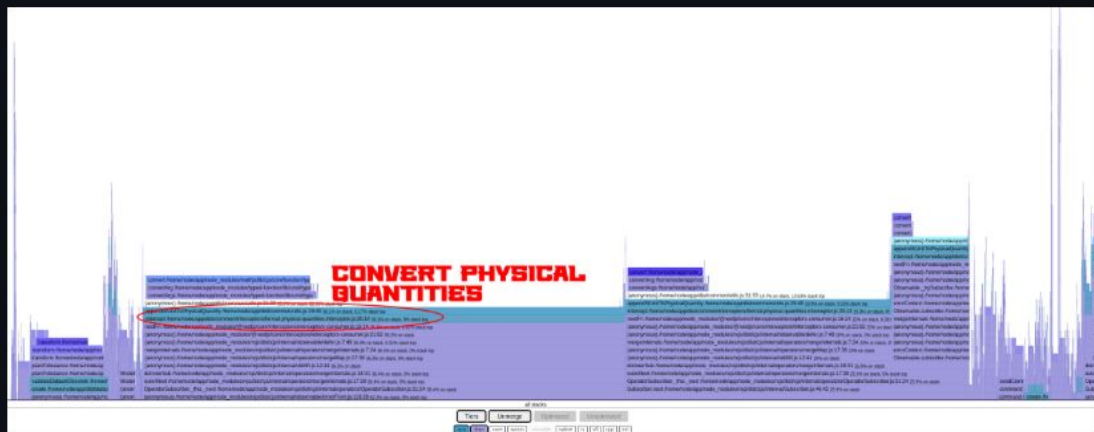
Ingvord on Mar 23

Member

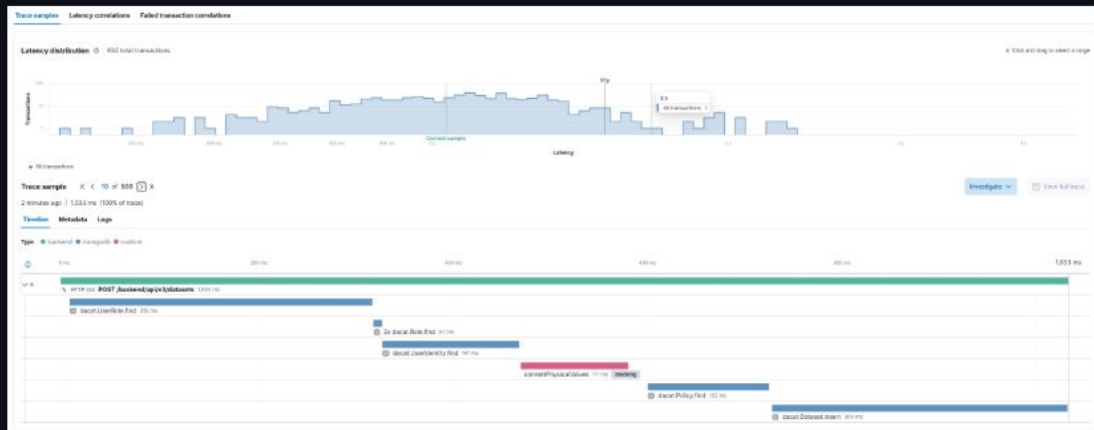
Author



Jo, I pinned down the suspect - it is `convert-physical-quantities-interceptor`. I this in the flame graph:



As well as in APM:



MORE DETAILS IN THE DAPHNE4NFDI ANNUAL MEETING POSTER

23



<https://zenodo.org/records/15068411>

zenodo Search records... Communities My dashboard ingvord.m...

DAPHNE4NFDI

Published March 22, 2025 | Version v1

Poster Open

Benchmarking SciCat Performance: Data-In & Data-Out for Beamline Operations .

Khokhriakov, Igor (Project leader)¹

Contributors

Others: Hinzmann, Regina¹; Parthasarathy Tirumalai Nallam Chakravarty¹

Show affiliations

Show affiliations

This work was partially supported by the consortium DAPHNE4NFDI in the context of the work of the NFDI e.V. The consortium is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - project number 460248799.

Files

SciCat_Benchmarking_Poster.pdf

1 of 1

Automatic Zoom

The poster content is displayed within a viewer window. It features a title slide with the text 'Data-In & Data-Out for Beamline Operations.' and authors 'Igor Khokhriakov, Parthasarathy Tirumalai Nallam Chakravarty (FS-SC) Regina Hinzmann (IT-INFA)'. The poster includes sections for 'Problem Statement', 'Results', and 'Methodology'. The 'Results' section contains several bar and line charts comparing SciCat performance across different datasets and request rates. The 'Methodology' section describes the experimental setup and data collection process.

4 VIEWS 16 DOWNLOADS

Show more details

Versions

Version v1 Mar 22, 2025

10.5281/zenodo.15068411

Cite all versions? You can cite all versions by using the DOI 10.5281/zenodo.15068410. This DOI represents all versions, and will always resolve to the latest one. Read more.

External resources

Indexed in

OpenAIRE

Communities

DAPHNE4NFDI

Details

DOI 10.5281/zenodo.15068411

Resource type Poster

Publisher Zenodo

Conference

SciCat, part 3

Data-In, Origdatablocks

Igor Khokhriakov aka Ingvord

OrigDatablocks

Uploading 1M DataFiles

```
Response Status: 413
Body: <html>
<head><title>413 Request Entity Too Large</title></head>
<body>
<center><h1>413 Request Entity Too Large</h1></center>
<hr><center>nginx/1.24.0 (Ubuntu)</center>
</body>
</html>
```

```
# client buffers
client_max_body_size 100M; # max allowed body size
client_body_buffer_size 10M;
client_header_buffer_size 16k;
# useful for big headers e.g. tokens, large queries etc
large_client_header_buffers 4 32k;
proxy_buffers 16 64K;
proxy_buffer_size 64k;
proxy_busy_buffers_size 128k;
proxy_read_timeout 1200;
proxy_send_timeout 60;

# prevent buffering large responses in memory
proxy_max_temp_file_size 0;

proxy_request_buffering off;
```

Error occurrence

3 minutes ago | [POST http://localhost/backend/api/v3/origdatablocks](#) | **500 Internal Server Error** | [POST unknown route](#)

Exception stack trace

Metadata

request entity too large

```
> at readStream (node_modules/raw-body/index.js:163)
> at getRawBody (node_modules/raw-body/index.js:116)
> at read (node_modules/body-parser/lib/read.js:79)
> at jsonParser (node_modules/body-parser/lib/types/json.js:138)
> at handle (node_modules/elastic-apm-node/lib/instrumentation/modules/express.js:80)
> at handle (node_modules/express/lib/router/layer.js:95)
> at trim_prefix (node_modules/express/lib/router/index.js:328)
> at <anonymous> (node_modules/express/lib/router/index.js:286)
> at process_params (node_modules/express/lib/router/index.js:346)
> at next (node_modules/express/lib/router/index.js:280)
> at arguments.<computed> (node_modules/elastic-apm-node/lib/instrumentation/modules/express.js:76)
> at cors (node_modules/cors/lib/index.js:188)
> at <anonymous> (node_modules/cors/lib/index.js:224)
> at originCallback (node_modules/cors/lib/index.js:214)
> at <anonymous> (node_modules/cors/lib/index.js:219)
> at optionsCallback (node_modules/cors/lib/index.js:199)
> at corsMiddleware (node_modules/cors/lib/index.js:204)
> at handle (node_modules/elastic-apm-node/lib/instrumentation/modules/express.js:80)
> at handle (node_modules/express/lib/router/layer.js:95)
> at trim_prefix (node_modules/express/lib/router/index.js:328)
> at <anonymous> (node_modules/express/lib/router/index.js:286)
> at process_params (node_modules/express/lib/router/index.js:346)
> at next (node_modules/express/lib/router/index.js:280)
> at expressInit (node_modules/express/lib/middleware/init.js:40)
> at handle (node_modules/elastic-apm-node/lib/instrumentation/modules/express.js:80)
> at handle (node_modules/express/lib/router/layer.js:95)
> at trim_prefix (node_modules/express/lib/router/index.js:328)
> at <anonymous> (node_modules/express/lib/router/index.js:286)
> at process_params (node_modules/express/lib/router/index.js:346)
> at next (node_modules/express/lib/router/index.js:280)
> at query (node_modules/express/lib/middleware/query.js:45)
> at handle (node_modules/elastic-apm-node/lib/instrumentation/modules/express.js:80)
> at handle (node_modules/express/lib/router/layer.js:95)
> at trim_prefix (node_modules/express/lib/router/index.js:328)
> at <anonymous> (node_modules/express/lib/router/index.js:286)
```

```
const config = {
  maxFileUploadSizeInMb: process.env.MAX_FILE_UPLOAD_SIZE || "16mb", // 16MB by default
```

OrigDatablocks

16MB Mongo document size limit

```
[Nest] 1 - 06/03/2025, 11:39:11 AM ERROR "Internal server error"
[Nest] 1 - 06/03/2025, 11:39:11 AM ERROR RangeError [ERR_OUT_OF_RANGE]: The value of "offset" is out of
[Nest] 1 - 06/03/2025, 11:39:11 AM ERROR Object:
{
  "requestUrl": "/backend/api/v3/origdatablocks",
  "requestUser": {
    "_id": "67d721c2948c9ce74a78d84a",
    "username": "admin",
    "email": "admin@scicat.project",
    "currentGroups": [
      "globalaccess",
      "admin"
    ],
    "authStrategy": "local"
  },
  "statusCode": 500
}
```

Number of DataFiles limit is ~75K

How to enter a document in mongodb with size larger than 16MB?

Working with Data Developer Tools



Sahildeep_Kaur

Apr 2022

Apr 2022

1/2

Apr 2022

Hi everyone,
I want to enter a document in mongodb with size larger than 16MB.
I have tried grids but it stores the data in binary. Is there any method by which we can store the data larger than 16MB in mongodb in object as I need to update the documents automatically. Thank you!

1 Reply



8.3k views 2 links



steevej Steeve Juneau

Apr 2022

Apr 2022



Sahildeep_Kaur:



Is there any method by which we can store the data larger than 16MB

None, see <https://www.mongodb.com/docs/manual/reference/limits/> 344 .

And thankfully because I feel there are real issue with your schema design if you end up with 16Mb structured documents to store. Media files, I understand, but structured data, I would like to ear about the use cases.

Can I increase document limit size in mongo collection?

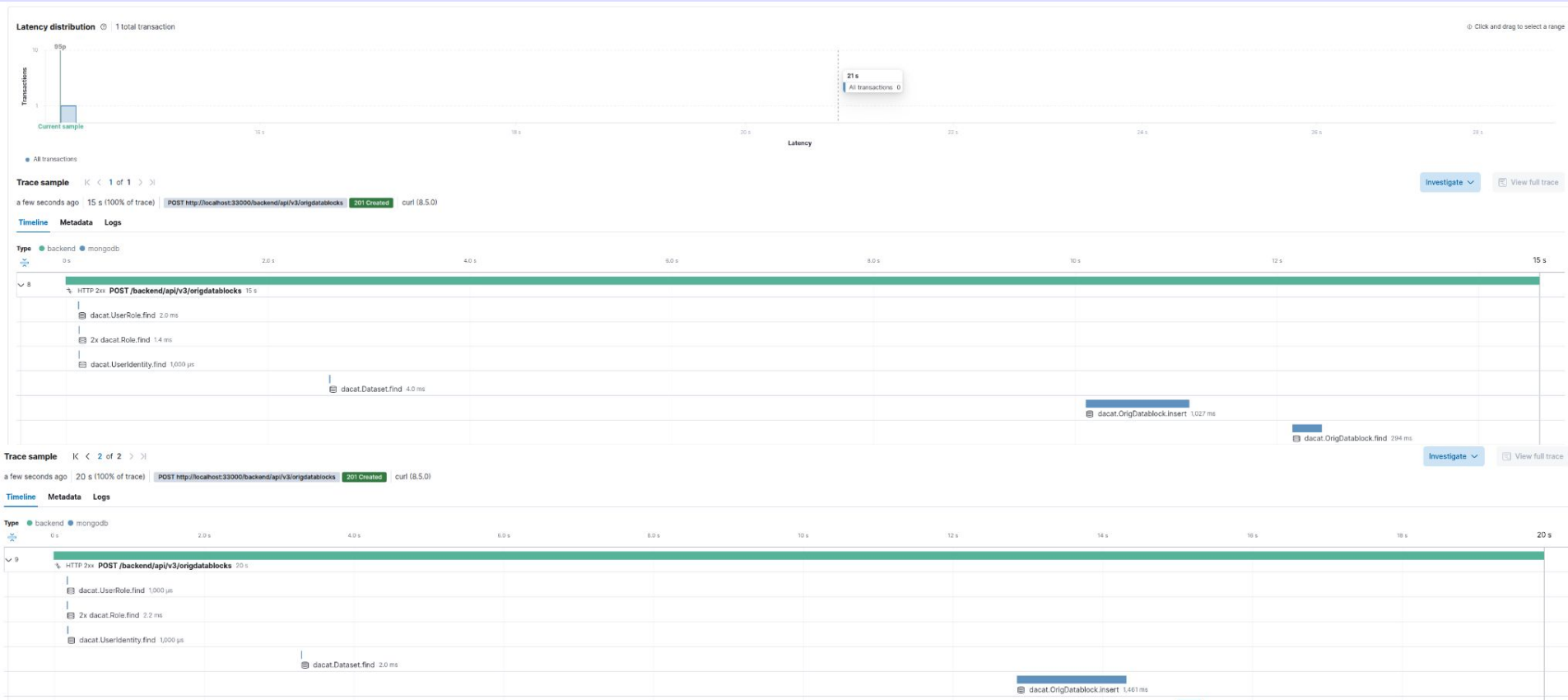
No, you **cannot increase the document size limit** in MongoDB — it's a **hard limit of 16MB** per document, and cannot be changed even through configuration or internal hacks.



Why?

- MongoDB is designed with a **16MB document limit** to ensure performance and memory efficiency.
- This limit applies to **any single BSON document**, including embedded arrays and subdocuments

OrigDatablocks



▼ 20

HTTP 2xx POST /backend/api/v3/origdatablocks 1,904 ms

OrigDatablocks

dacadat.Dataset.create 10 ms

dacadat.UserRole.find 13 ms

dacadat.Dataset.createIndexes 7.0 ms

2x dacat.Role.find 4.5 ms

dacadat.Dataset.createIndexes 6.0 ms

dacadat.UserIdentity.find 2.0 ms

5x dacat.Dataset.createIndexes 1,435 ms

Single (curl) 1K records ingestion, fresh Mongo

▼ 1

await dataset 68 ms

dacadat.Dataset.find 9.0 ms

▼ 2

await create 1,104 ms

▼ 1

OrigDatablocksService.create 294 ms blocking

dacadat.OrigDatablock.insert 146 ms

▼ 3

updates datasets size 219 ms

OrigDatablocks

Single (curl) 75K records ingestion, fresh Mongo

HTTP 2xx POST /backend/api/v3/origdatablocks 21 s

dacat.Dataset.create 3.0 ms

dacat.Role.find 5.0 ms

dacat.Dataset.createIndexes 9.0 ms

2x dacat.Role.find 12 ms

dacat.Dataset.createIndexes 12 ms

dacat.UserIdentity.find 3.0 ms

17x dacat.Dataset.createIndexes 14 s

await dataset 24 ms

dacat.Dataset.find 10 ms

await create 15 s

OrigDatablocksService.create 3,568 ms blocking

dacat.OrigDatablock.insert 1,958 ms

updates datasets size 2,358 ms

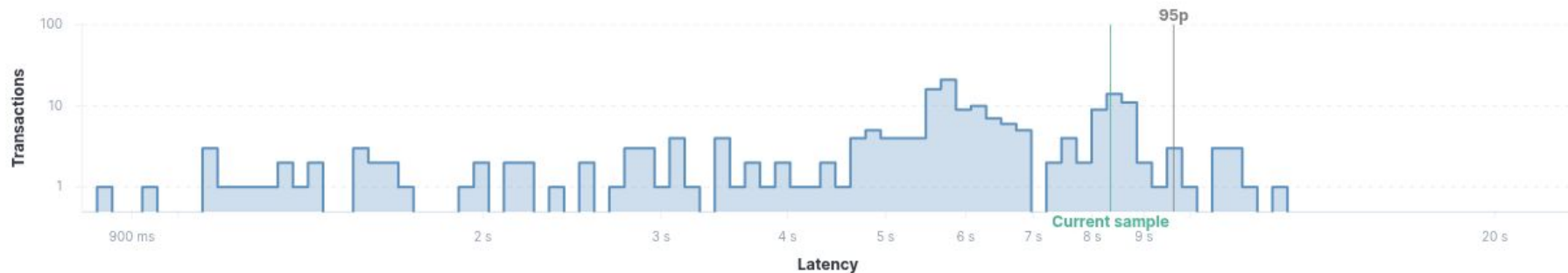
dacat.OrigDatablock.find 514 ms

OrigDatablocks

1K records ingestion (-c10 -t10 -R10)

Latency distribution ? 212 total transactions

Click and drag to select a range



● All transactions

Trace sample K < 5 of 213 > X

Investigate ▾

View full trace

a minute ago 8,342 ms (100% of trace)

POST http://localhost/backend/api/v3/origdatablocks

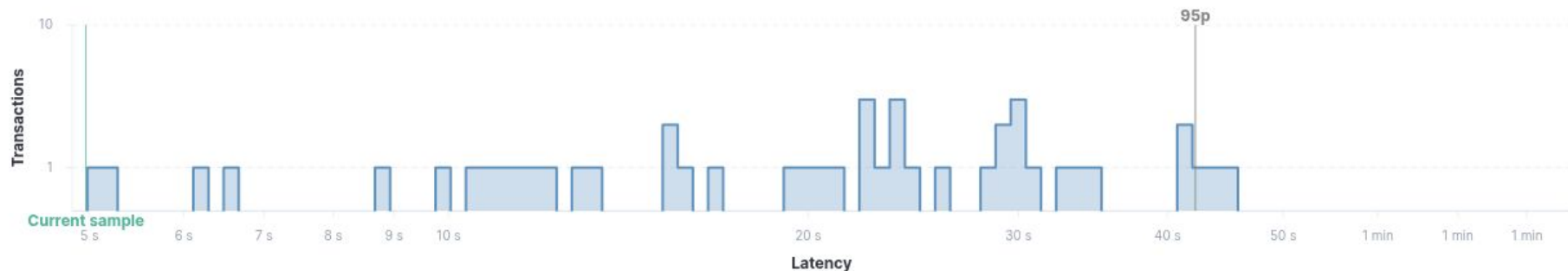
201 Created

OrigDatablocks

10K records ingestion (-c10 -t10 -R10)

Latency distribution ⓘ 46 total transactions

ⓘ Click and drag to select a range



● All transactions

Trace sample K < 55 of 55 > >|

Investigate ▾

View full trace

4 minutes ago | 4,966 ms (100% of trace) |

POST http://localhost/backend/api/v3/origdatablocks

201 Created

Timeline Metadata Logs

Type ● backend ● mongodb ● custom



0 ms

1,000 ms

2,000 ms

3,000 ms

4,000 ms

4,966 ms

5

OrigDatablocks

Latency distribution ⓘ 10 total transactions

① Click and drag to select a range



● All transactions

Trace sample ⏪ < 1 of 11 > ⏩

Investigate ▾

📄 View full trace

2 minutes ago | 86 s (100% of trace) | POST http://localhost/backend/api/v3/origdatablocks 201 Created

Timeline Metadata Logs

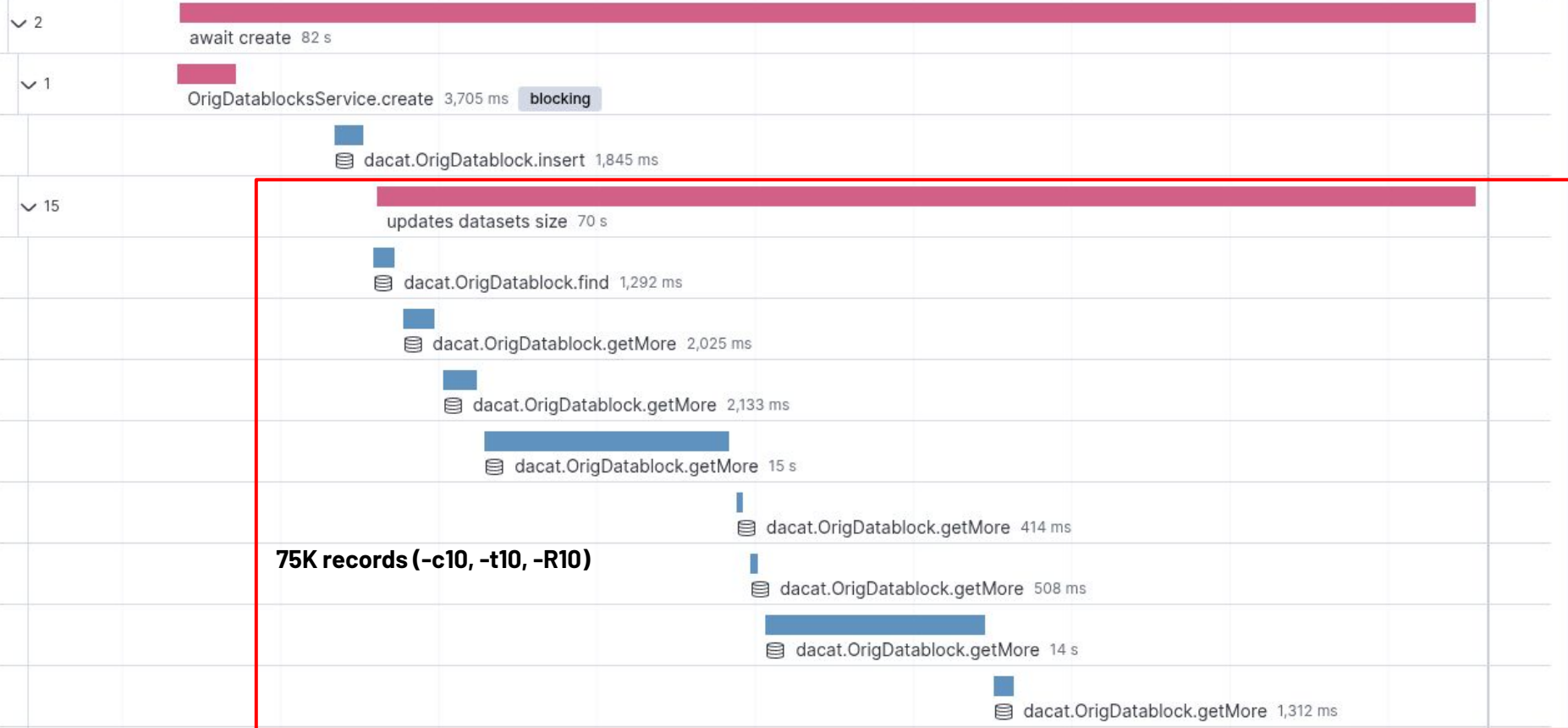
Type ● backend ● mongodb ● custom



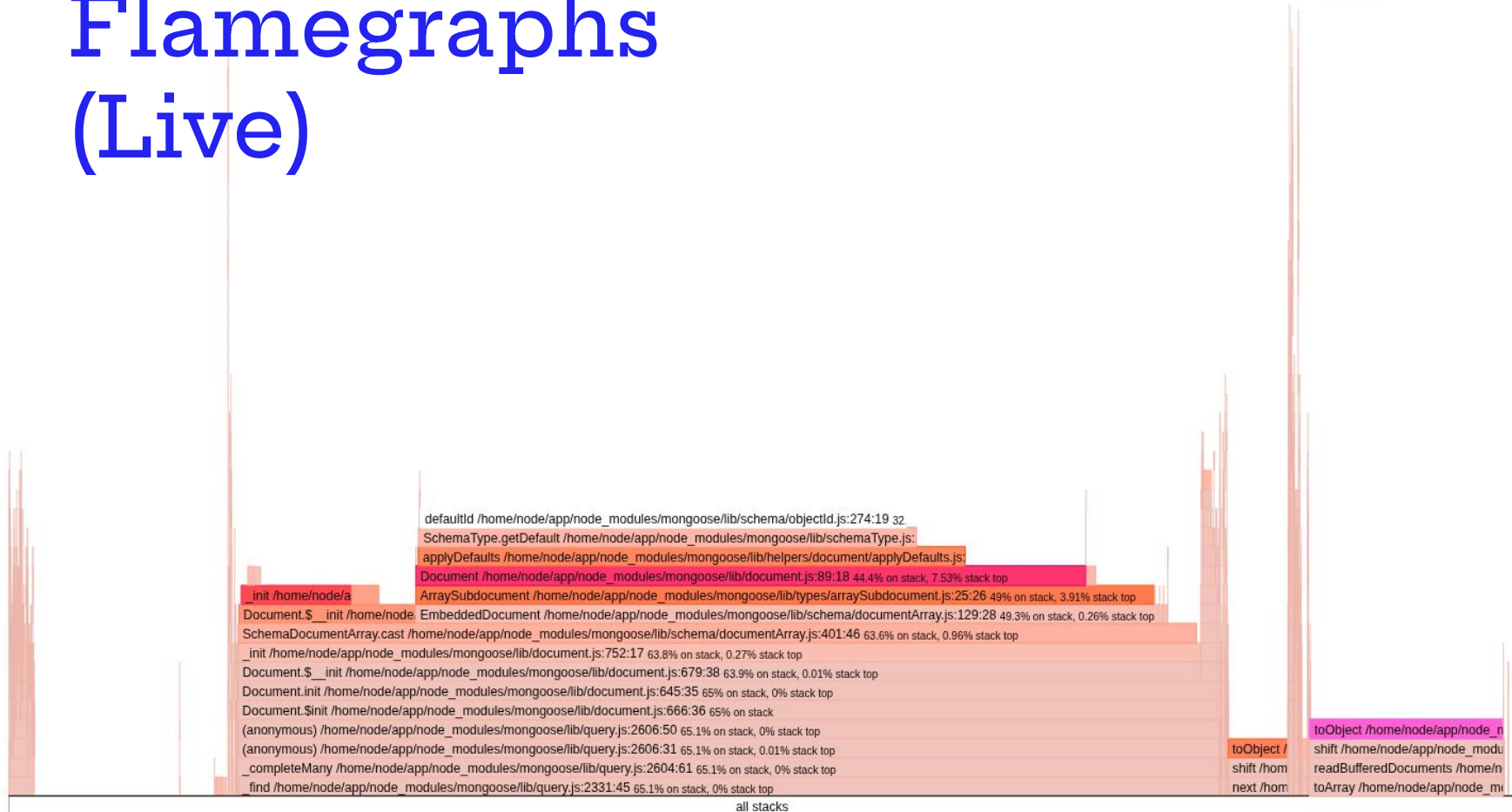
0 s 10 s 20 s 30 s 40 s 50 s 60 s 70 s 80 s 86 s

6 success POST /backend/api/v3/origdatablocks 86 s

OrigDatablocks



Flamegraphs (Live)



<https://ingvord.github.io/animated-garbanzo>

```
213   async updateDatasetSizeAndFiles(pid: string) : Promise<void> { Show usages  Max Novelli +1
214     // updates datasets size
215     const parsedFilters: IFilters<OrigDatablockDocument, IOrigDatablockFields> =
216     { where: { datasetId: pid } };
217     const datasetOrigdatablocks: OrigDatablock[] =
218     await this.origDatablocksService.findAll(parsedFilters);
219
220     const updateDatasetDto: PartialUpdateDatasetDto = {
221       size: datasetOrigdatablocks
222         .map((od : OrigDatablock ) => od.size)
223         .reduce((ps : number , a : number ) => ps + a, 0),
224       numberOfFiles: datasetOrigdatablocks
225         .map((od : OrigDatablock ) => od.dataFileList.length)
226         .reduce((ps : number , a : number ) => ps + a, 0),
227     };
```

- MongoDB `getMore` feeds batches of documents to the app (APM)
- Mongoose applies defaults + initializes models for each document fetched (flamegraph)
- Together: high latency + CPU cost arise from fetching + transforming large document sets
- Optimization target: use aggregate or lean queries (<https://mongoosejs.com/docs/tutorials/lean.html>)

AVAILABLE FOR HIRE STARTING IN SEP'25



Igor Khokhriakov

Hamburg, Hamburg, Germany · [Contact Info](#)

3K followers · 500+ connections



[See your mutual connections](#)



DES Y



**Russian State University of
Humanities**



Websites

K8S realm
4 replicas
2 mongoDb rs

1_000_000 Records, 10 Clients, Duration 120s, Rate {10..1000}

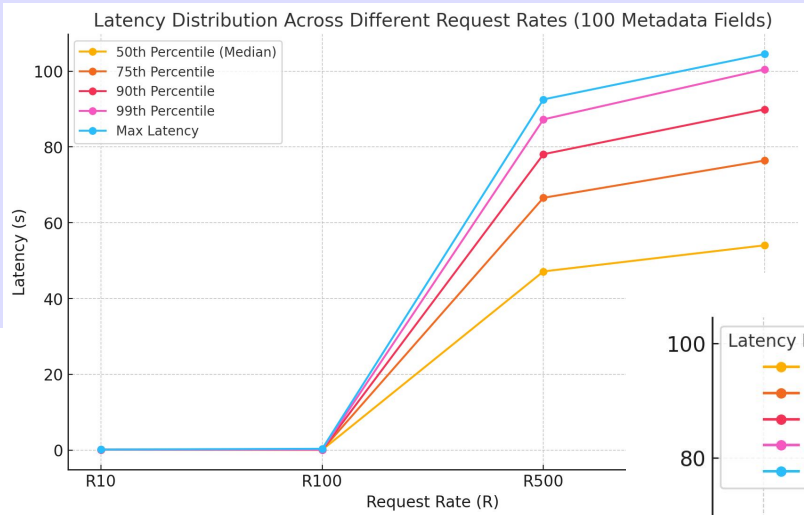
Local VM VS K8s setup
Data-out, PID (w/index)

Configuration	Avg Latency	50th %ile Latency	99th %ile Latency	Requests/sec	Timeouts
R1000	127.60ms	6.82ms	2.08s	997.52	-
R100	14.49ms	11.60ms	74.30ms	100.06	-
R10	17.87ms	15.08ms	148.48ms	10.02	-

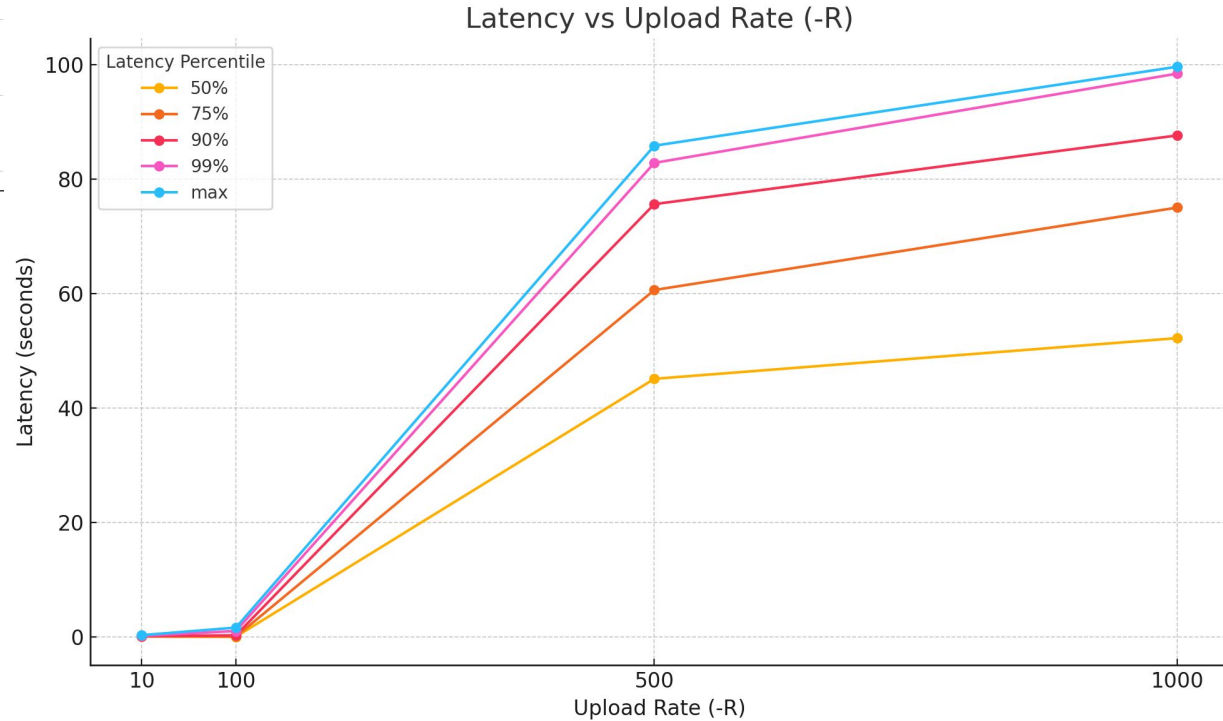
-R	Total Requests	Avg Latency	50% Latency	99% Latency	Max Latency	Req/sec
1000	120000	47.72ms	6.47ms	786.94ms	1.03s	999.97
100	12010	9.01ms	8.60ms	15.44ms	29.90ms	100.07
10	1209	15.96ms	12.90ms	32.96ms	70.85ms	10.07

K8s setup significantly
better at 99% percentile
(mongoDb rs?!)

Local VM VS K8s setup Data-In, 100 metadata

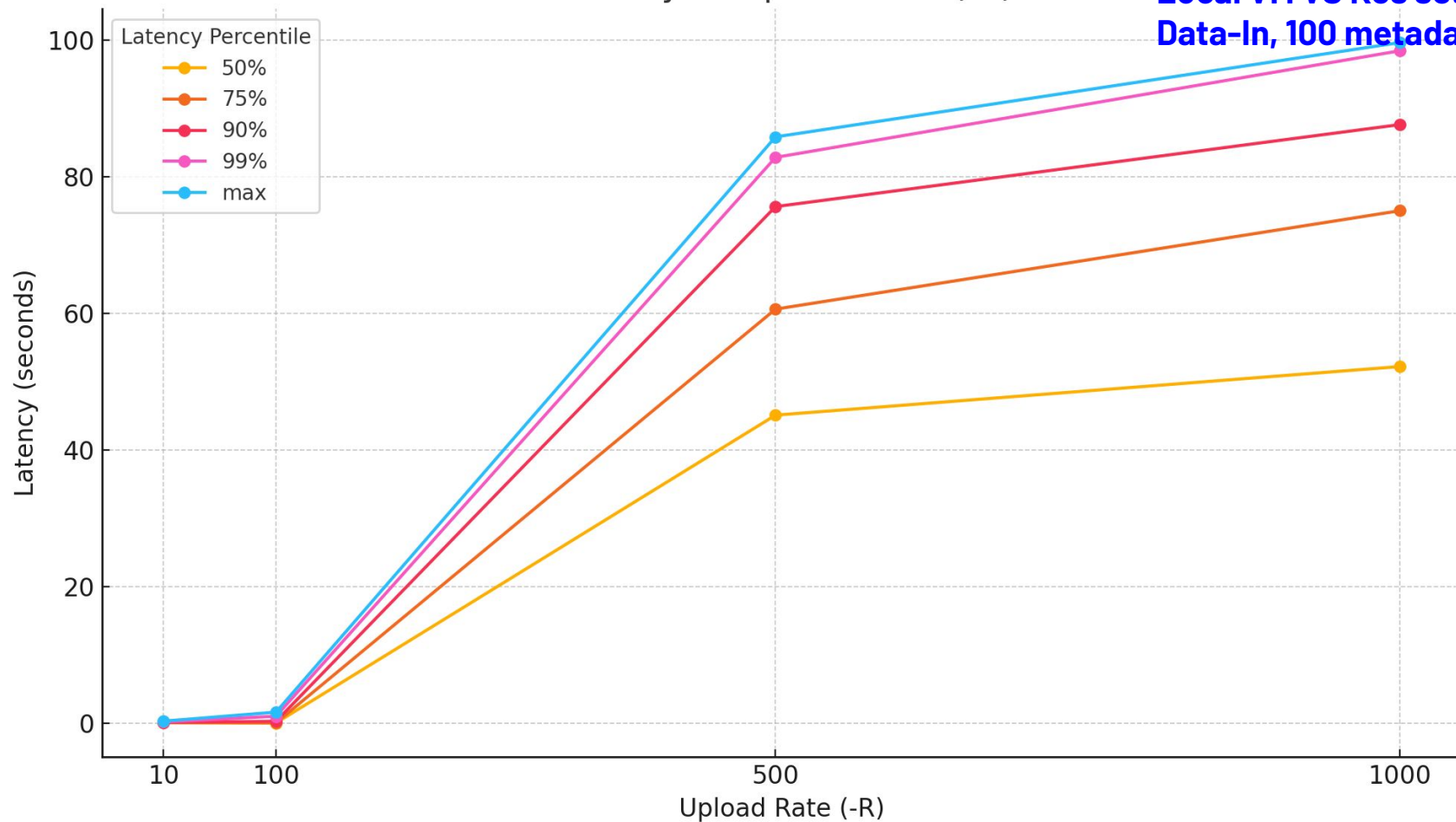


**Yes, mostly identical
(as expected - due to application
code)**



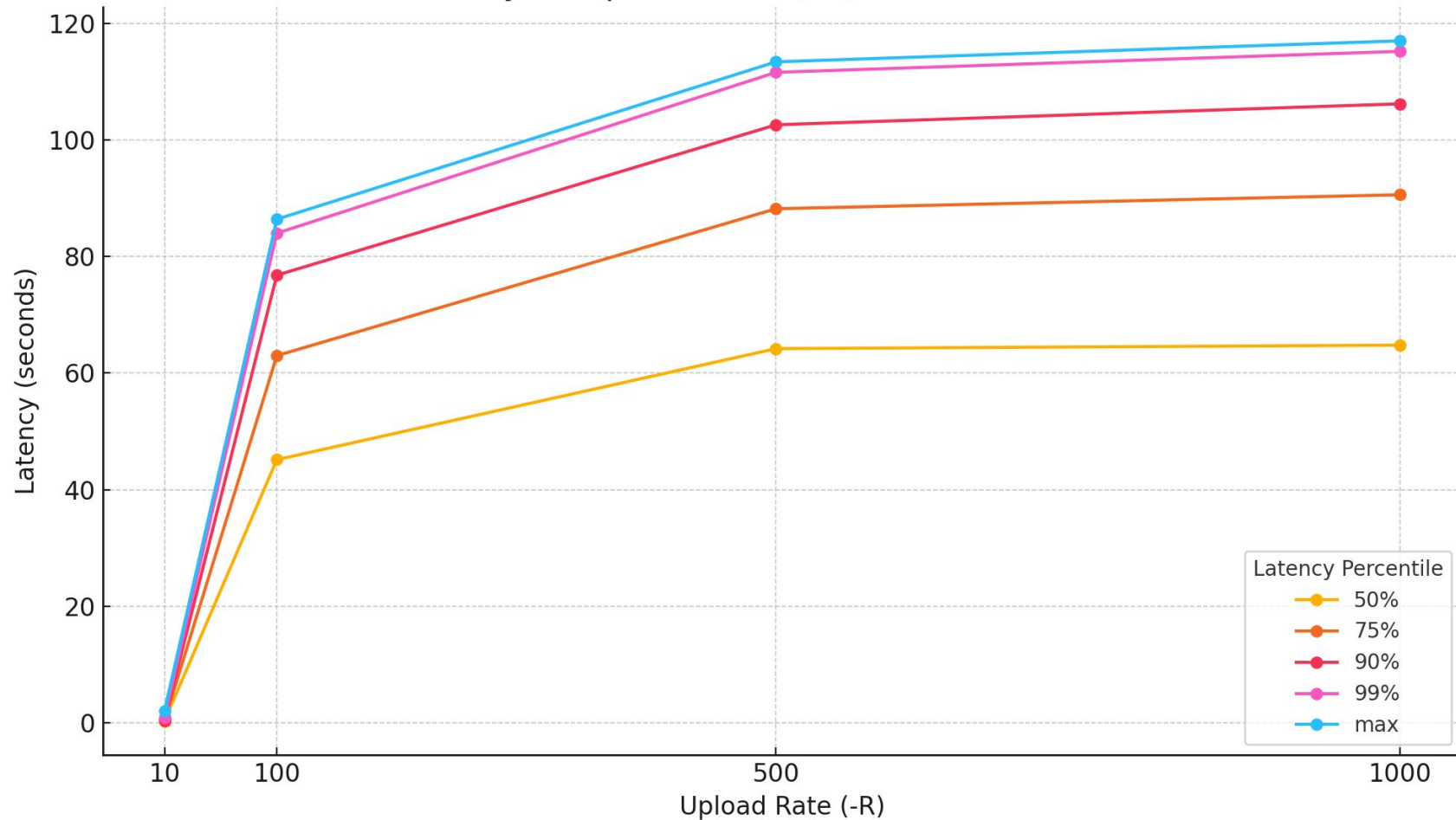
Latency vs Upload Rate (-R)

Local VM VS K8s setup
Data-In, 100 metadata

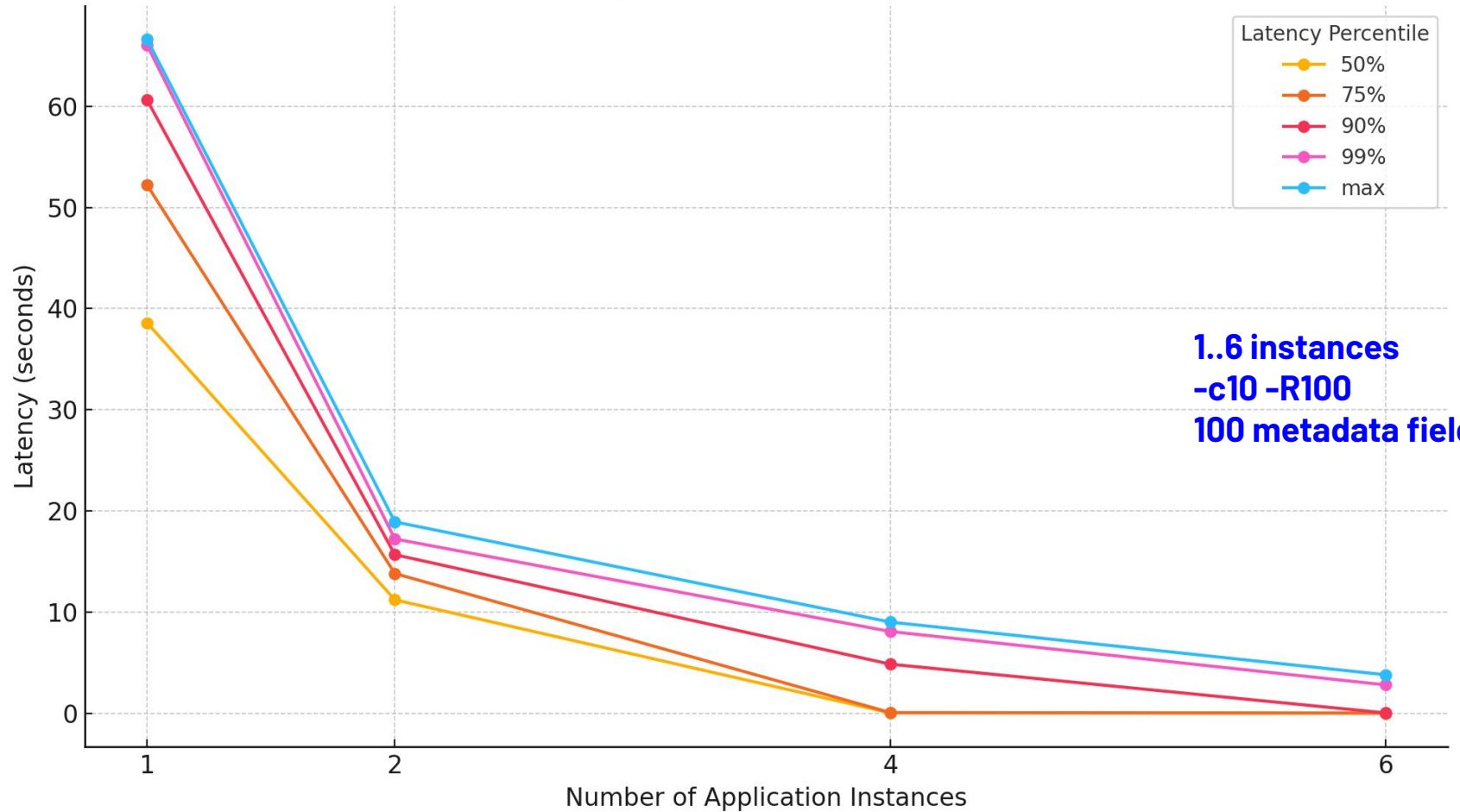


Latency vs Upload Rate (-R) — Second Test Set

1K metadata

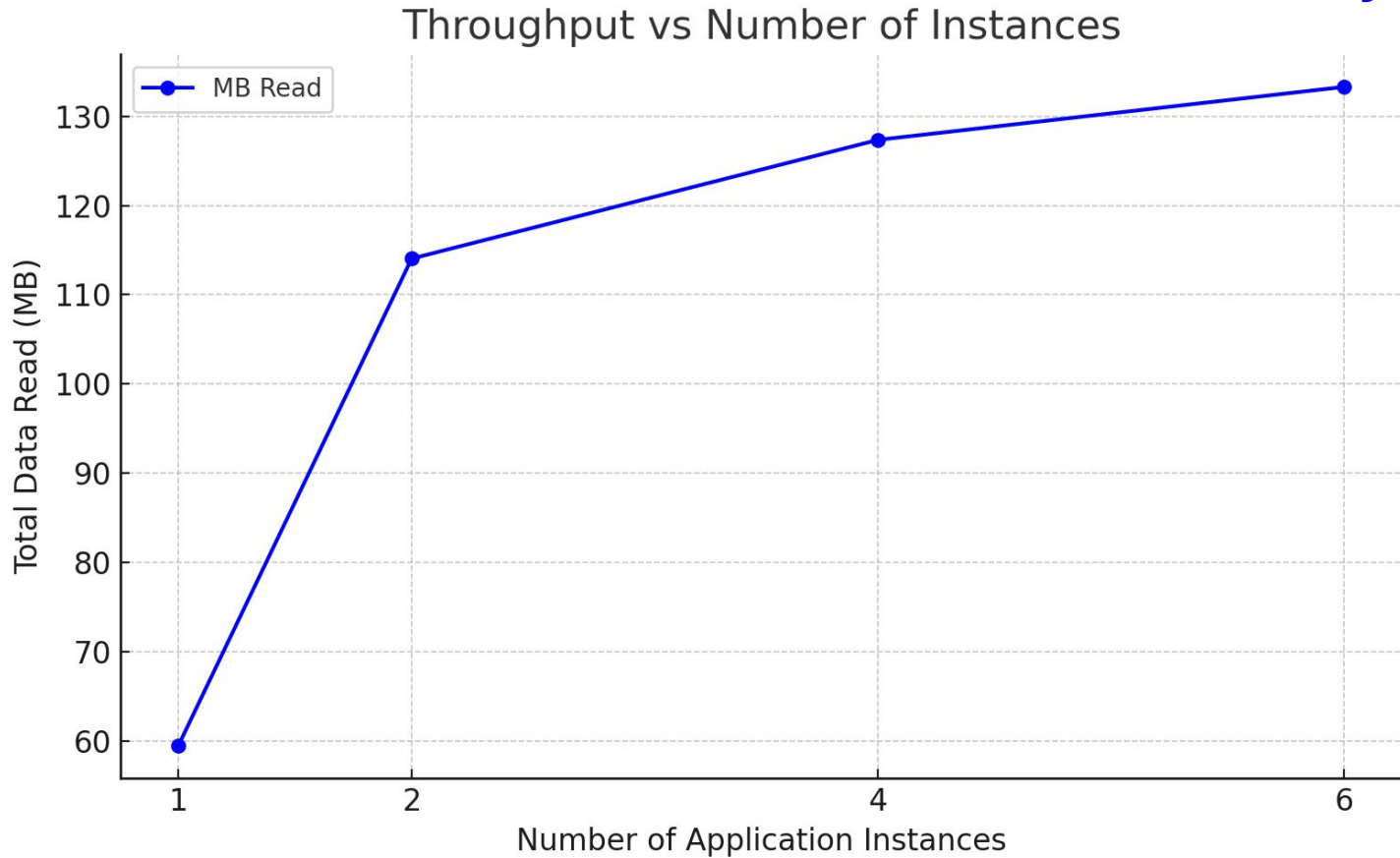


Latency vs Number of Instances



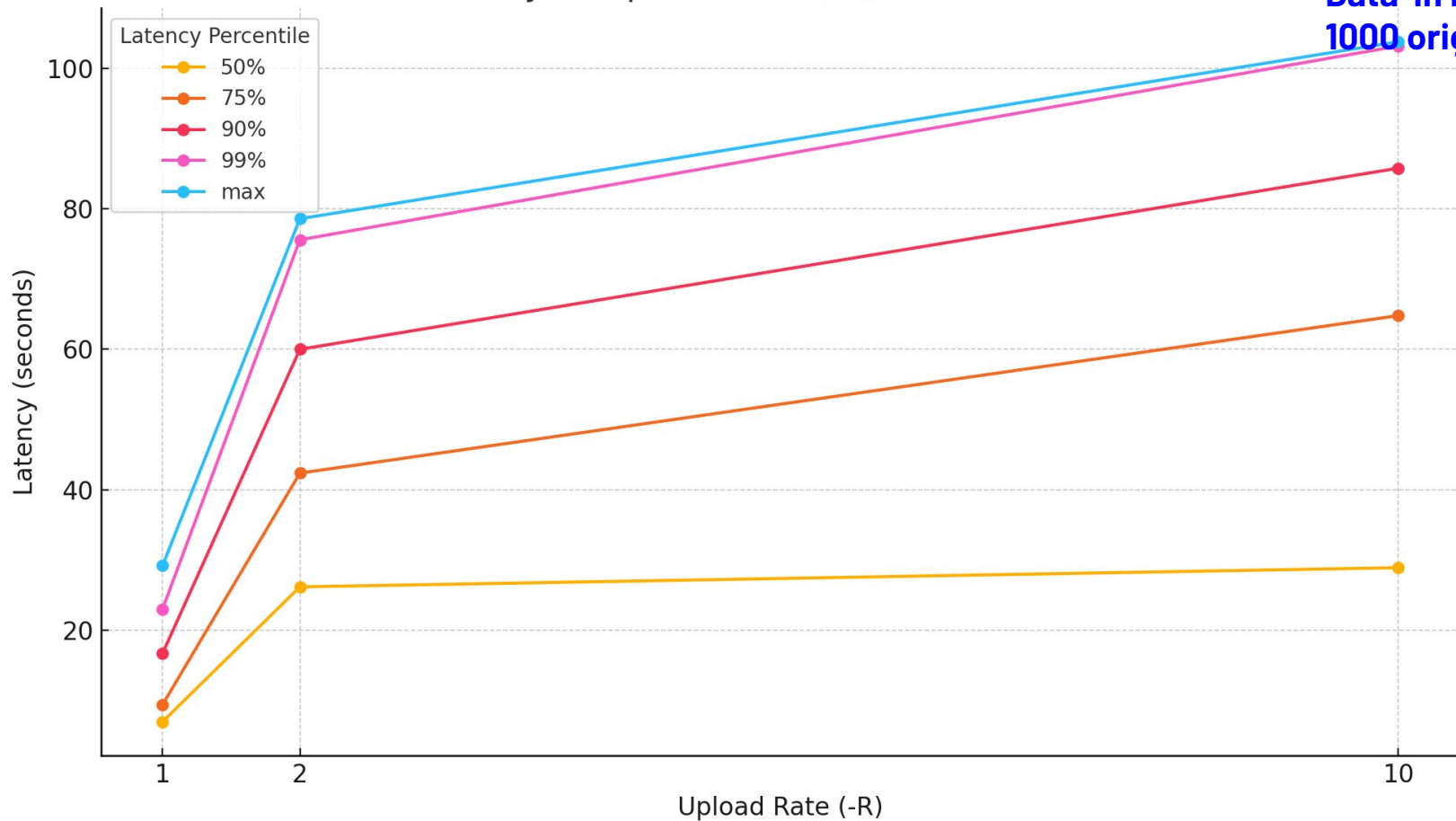
1.6 instances
-c10 -R100
100 metadata fields

100 metadata
throughput increase



g elit.

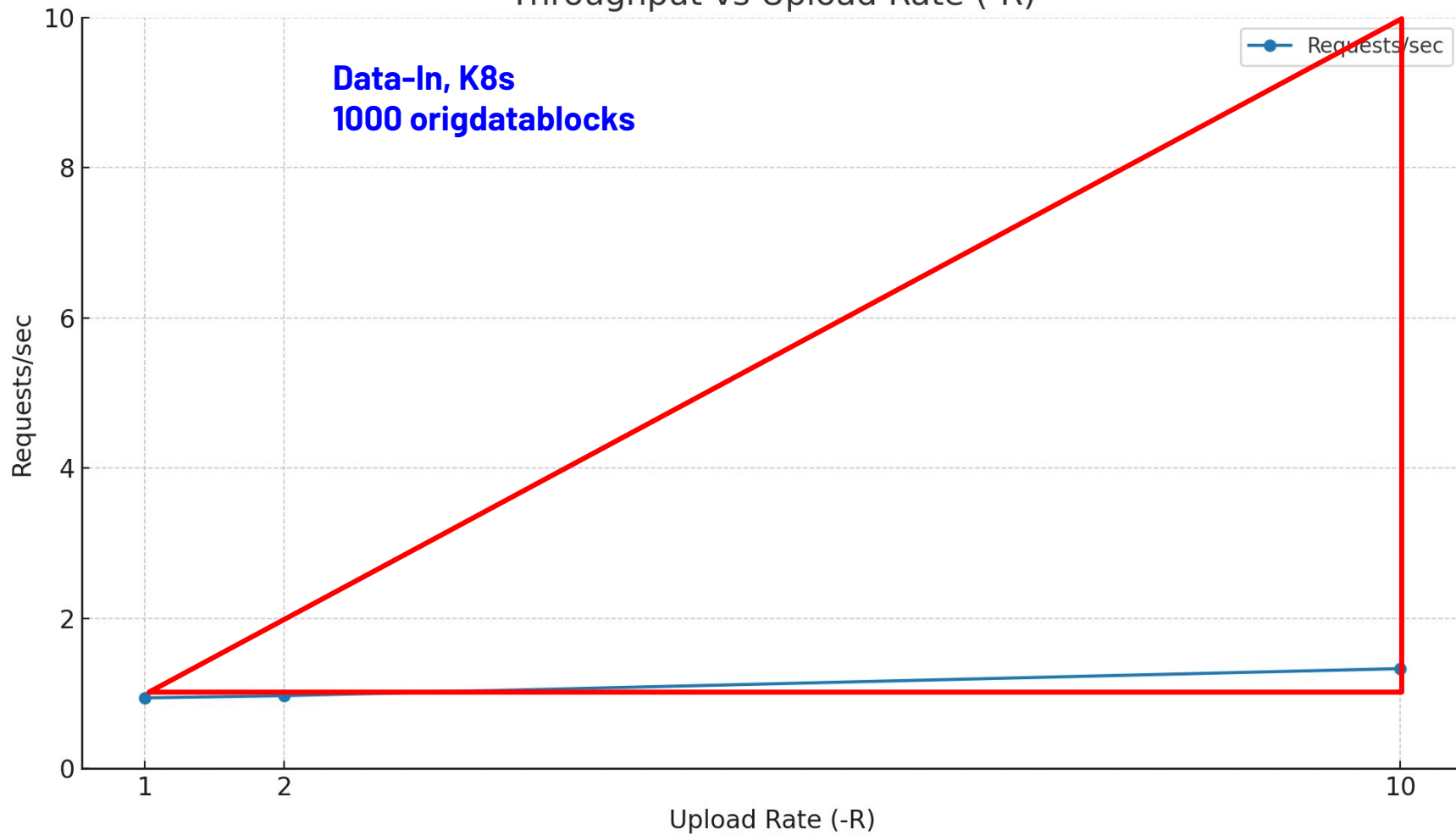
Latency vs Upload Rate (-R) — Final Test Set



Data-In K8s
1000 origdatablocks

elit.

Throughput vs Upload Rate (-R)



SIDE NOTES

```
raw 2025-07-04T21:19:31.496+0200 [#####.] dacat.Dataset 3.56GB/3.58GB (99.5%)
raw 2025-07-04T21:19:34.496+0200 [#####.] dacat.Dataset 3.57GB/3.58GB (99.8%)
raw 2025-07-04T21:19:36.113+0200 [#####.] dacat.Dataset 3.58GB/3.58GB (100.0%)
raw 2025-07-04T21:19:36.113+0200 1000001 document(s) imported successfully. 0 document(s) failed to import.
raw khokhria@xenv2:~/animated-garbanzo$ mongosh mongodb://dacatuser:dacatpass@localhost:27117/dacat
Current Mongosh Log ID: 68682aa235d9c8300e567a2a
derive Connecting to: mongodb://<credentials>@localhost:27117/dacat?directConnection=true&serverSelectionTimeoutMS=200
derive Using MongoDB: 7.0.14
derive Using Mongosh: 2.3.7
derive mongosh 2.5.5 is available for download: https://www.mongodb.com/try/download/shell
For mongosh info see: https://www.mongodb.com/docs/mongosh-shell/
raw rs0 [direct: primary] dacat> db.Dataset.countDocuments()
1111961
raw rs0 [direct: primary] dacat> █
```

Items per page:

25 ▼

1 - 25 of 499265

<

>

Thanks

Directed by
Igor Khokhriakov aka Ingvord